EOSC-SYNERGY

EU DELIVERABLE: D3.1

Software Maturity baseline

Document Identifier:	EOSC-SYNERGY-D3.1
Date:	29/06/2020
Activity:	WP3
Lead Partner:	LIP
Document Status:	APPROVED
Dissemination Level:	PUBLIC
Document Link:	

https://drive.google.com/file/d/1Ac5GEnghN3afNDLvHMFElDatgrP7X0jf/

Abstract:

This deliverable describes the quality requirements and best practices to be considered when validating software for EOSC services within EOSC-Synergy. The document also describes the badge issuing process as a reward mechanism for compliance towards quality.

EOSC-SYNERGY receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 857647

I. Copyright Notice

II. Delivery Slip

	Name	Partner/Activity	Date
From	Mário David	LIP/WP3	22/06/2020
Reviewed by	**Moderator:** Isabel Campos **Reviewers:** Valentin Kozlov, Amanda Calatrava	CSIC/WP1 KIT/WP2 UPV/WP4	26/06/2020
Approved by	PMB	PO	29/06/2020

III. Document Log

Issue	Date	Comment	Author/Partner
v1	11/05/2020	TOC and initial draft	J.Gomes/LIP
v2	01/06/2020	Added Quality Badges section Added badges issuance	Germán Molto/UPV, Miguel Caballer/UPV
v3	10/06/2020	Added software & services Quality Added Criteria Validation and Verification Added SQAaaS architecture Added appendixes	Mário David/LIP Pablo Orviz/CSIC Samuel Bernardo/LIP João Pina/LIP Vyacheslav Tykhonov/DANS Jorge Gomes/LIP
v4	22/06/2020	Executive summary, Introduction, conclusions, layout, and revision	Jorge Gomes/LIP Mário David/LIP

IV. List of Acronyms

Acronym	Description
API	Application Programming Interface
CAMS	Culture, Automation, Measurement and Sharing

CD	Continuous Delivery
CI	Continuous Integration
CLI	Command Line Interface
COP	Container Orchestration Platform
COTS	Commercial Off The Shelf
DAST	Dynamic Application Security Testing
DOI	Digital Object Identifier
EOSC	European Open Science Cloud
IaC	Infrastructure as Code
JSON	JavaScript Object Notation
JWS	JSON Web Signature
MD	Markdown
OWASP	Open Web Application Security Project
OSS	Open Source Software
PDF	Portable Document Format
PID	Persistent Identifier
PR	Pull Request
SaC	Security as Code
SAST	Static Application Security Testing
SCM	Software Configuration Management
SDLC	Software Development Life Cycle
SMM	CESSDA's Software Maturity Model
SMS	Service Management System
SQA	Software Quality Assurance (also including Services Quality Assurance)
TRL	Technology Readiness Level
VCS	Version Control System
WP	Work Package

Table of Contents

1 Introduction **7**
1.1 Notational Conventions 8

2 Quality Criteria Development Methodology **9**
2.1 State of the Art 9
2.2 Document management 11
2.3 Metrics of the GitHub repositories 11
2.4 Synergies with other projects 12
2.5 Use cases 13

3 Software Quality Criteria **14**
3.1 Purpose and Goals 14
3.2 Overview of the Software Quality Criteria 15

4 Services Software Criteria **17**
4.1 Purpose and Goals 18
4.2 Contextualization of a Service 18
4.3 Overview of the Services Quality Criteria 19

5 Criteria Validation and Verification **23**
5.1 The SQA as a Service (SQAaaS) 23
5.2 SQAaaS: initial steps 23
5.2.1 Main outcomes 23
5.2.2 Architecture 24

6 Quality Badges **26**
6.1 Purpose 26
6.2 State of the Art 26
6.2.1 Open Badges: Specification for Digital Badges 26
6.2.2 Open-Source Tools to Issue Open Badges 28
6.3 Badgr Deployment in EOSC-Synergy 29
6.4 Badges Graphic Design 30

7 Conclusions **31**

8 References **32**

9 Annex I - Badges Issuance with Badgr **34**
9.1 Issuing Badges through the GUI 34

9.2 Issuing Badges through the API 37

10 Annex II - Docker related files for Badgr **40**

11 Annex III - List of Designed Badges **43**

Executive Summary

Quality understood as reliability, sustainability, and reusability, is a fundamental aspect for a successful uptake of EOSC services by the European research communities. EOSC-Synergy is developing a quality based approach to foster the adoption of EOSC services, which will improve, promote and reward quality.

Two sets of quality baseline criteria have been established addressing both software quality, section 3, and services quality, section 4. The software quality criteria are based on widely adopted best practices and first-hand experiences that have been collected and successfully applied in previous projects such as INDIGO-DataCloud, DEEP-HybridDataCloud and eXtreme DataCloud. This set of quality criteria has been further developed by EOSC-Synergy taking into account its application to the EOSC environment.

The services quality criteria is a new development initiated by EOSC-Synergy. It is composed of a set of best practices aiming at ensuring common coherent quality attributes for services. While the software criteria is focused on software development, the services criteria is focused on deployment and delivery.

The two sets of quality baseline criteria are publicly available at the EOSC-Synergy organization in GitHub to foster external contributions. Through this collaborative process, the quality baseline criteria will be continuously improved and further detailed thus ensuring its maintenance during the project and beyond. The quality criteria already benefited from contributions and interactions with FAIRsFAIR and CESSDA.

The quality baseline criteria is the basis for the EOSC-Synergy software quality as a service (SQAaaS), which is being currently developed and whose architecture fundamentals are described in section 5.

Adherence to the quality baseline criteria will be rewarded through quality badges, see section 6. These badges can be added to software repositories and web objects to increase visibility and provide a verifiable method to assess the quality achievements of the services. The increased visibility and trustworthiness provided by this certification process will contribute to increase the adoption of the EOSC services.

1. Introduction

Quality assessment is an important trait for software and for services. It allows users and managers to have higher trust that during its use and operation, the software and related services will work as supposed, give the expected results and meet their requirements. Furthermore, it also contributes to the maintainability, stability and sustainability of the software and services. Finally, it contributes to facilitating the collaboration between software developers and promotes good practices of software development.

Issuing badges as a result of a successful Quality Criteria verification process, is not only a visual or graphical way to show users and service managers the quality of the software and services, but as well, a way to expose the details of that assessment.

The set of criteria being developed by the project is described in the documents "*Software Quality Assurance baseline document*" [R6] and "*Service Quality Assurance baseline document*" [R12], do not follow fully or exclusively any of the Quality models reviewed in section 2.1 "State of the Art", although there are criteria which is similar or almost the same as some of the criteria or characteristics of those models. On the one hand, several of those Quality models had closed/commercial software in mind, some do not have tools for automated verification, or measurement of the characteristics or metrics. The ones which have tools allowing automatization of the measurement of metrics may or may not be open source.

The set of Quality criteria described herein has the following objectives in mind:

- Follow a DevOps pragmatic approach:
  - Include best practices of Software development, or service operation.
  - Pursue automation of criteria verification.
  - Enable a criteria verification flow through a pipeline, as a consequence of Continuous Integration and Continuous Delivery (CI/CD) process.
- Decouple the criteria from the tools and technologies used for validation providing abstraction and enabling developers, service managers and/or infrastructure operators to choose which tools and technologies to use for each purpose.
- Rewarding criteria conformance (e.g. through badges or other means).

To achieve the goal of "Higher Quality of Services to be integrated in EOSC", the workflow shown in Figure 1 is being designed and implemented in the EOSC-Synergy project.

Starting at the Software Quality Assurance phase (on the left of Figure 1), the criteria described in the "*Software Quality Assurance baseline document*" [R6] is applied through the use of a (CI/CD) pipeline. Once the quality verifications are successfully performed, the final

steps will consist in building the artefacts and requesting a Badge attesting its quality. As previously said, the badge contains a report on the quality assessment of the component.

The second phase is the Service Quality Assurance (on the right of Figure 1). Upon service deployment, the criteria described in the "*Service Quality Assurance baseline document*" [R12], is applied through the use of a CI/CD pipeline. Again, once the verifications are successfully performed, the final step in the pipeline will be requesting a Badge attesting its quality and certifying its suitability for integration in the EOSC infrastructure.

The main tool being used for the CI/CD pipelines is Jenkins CI service. Within the EOSC-Synergy project, Jenkins pipelines are under development for the project's Thematic Services. A further main objective of the project, is to develop and provide a Software Quality Assurance as a Service (SQAaaS), so that any software developer team or any service infrastructure operator can compose its own pipeline on demand, with whatever nodes they deem appropriate.

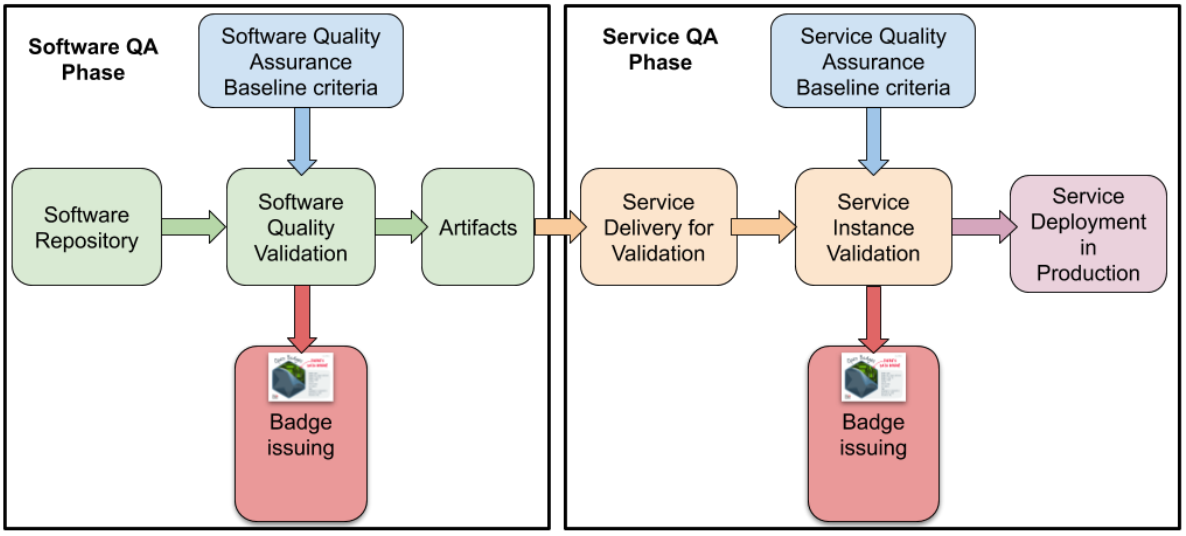

Figure 1: Quality assurance workflow architecture.

1.1 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [R15].

2. Quality Criteria Development Methodology

The EOSC-Synergy Quality Assurance process for Software and Services features a set of quality criteria based on best practices following the DevOps approach. The criteria is meant to be verified in an automated way, and is agnostic with respect to the technologies or services used to verify it.

The criteria is binary, i.e., either it is verified successfully or not. In the case of Software, the criteria is agnostic to programming languages, systems where the code is hosted, and places where the documentation is published among others. While in the case of Services, it is agnostic to the environment, hardware, underlying Operating System, and hosting systems i.e., bare-metal machines or virtualized resources such as containers and hypervisors.

2.1 State of the Art

The most relevant quality model [R31], is the standard defined in the ISO/IEC 25010:2011(en) [R29] Systems and software engineering, denoted: *"Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models"*. It replaces the ISO/IEC 9126-1:2001. The ISO/IEC 25010:2011(en) defines the following two models:

a) **A quality in use** model composed of five characteristics (some of which are further subdivided into sub-characteristics) that relate to the outcome of interaction when a product is used in a particular context of use. The five main characteristics are:1) **Effectiveness**, 2) **Efficiency**, 3) **Satisfaction**, 4) **Freedom from risk** and 5) **Context coverage**.

b) **A product quality** model composed of eight characteristics (which are further subdivided into sub-characteristics) that relate to static properties of software and dynamic properties of the computer system. The eight characteristics are: 1) **Functional suitability**, 2) **Performance efficiency**, 3) **Compatibility**, 4) **Usability**, 5) **Reliability**, 6) **Security**, 7) **Maintainability** and 8) **Portability**.

The **Quality in Use** model therefore seeks to quantify the "usability" (effectiveness, efficiency and satisfaction) of the application, when specific users attempt to meet their specified goals [R30], and freedom from risk as the "degree to which a product or system mitigates the potential risk to economic status, human life, health, or the environment" [R29]. The quantification of the characteristics of this model, are user based.

The **Product Quality** model characteristics are of more interest to the developers of the product (or Software component). The implementation and documentation of "plans" to quantify and verify each of the characteristics, are the responsibility of the developers or "supplier" of the product.

On the one hand both the **Quality in Use** and the **Product Quality** models only tackle "external quality" and require the Software in execution within a given environment. On the other hand, only some of the **Product Quality** model characteristics are appropriate for automation. In either case, there are no "internal quality" characteristics, which are related to static code of the software. The basis for the standard was connected to Commercial Off The Shelf (COTS) software.

The Open Source Software (OSS) has characteristics that are not present in COTS, such as public access to the source code and participation of community members (both the development and user side). Several such models are reviewed in [R32]. Some have their origin in the ISO/IEC 9126-1:2001, some are hierarchical and some are based on a Maturity model. The metrics corresponding to the assessment of the quality, are in general based on a given algorithm implemented in tools that automate the assessment of the quality.

The DevOps approach links development and operations for software components, through the use of a Continuous Integration and Continuous Delivery pipeline. The CAMS model stands for Culture, Automation, Measurement and Sharing, which are named as the four-fundamental dimensions to enable DevOps [R33]. It does not have a single standard, but takes the best practices from several standards. The authors of [R34], propose a Quality model based on DevOps, while it reviews some of the other Quality models. The metrics are based on several sources and are posed as questions. These questions/metrics focus on the number of features delivered, the time a feature needs to be delivered or the number of releases to deliver these features. They also map most of the metrics with the Product Quality model of the ISO/IEC 25010:2011.

Regarding Quality models in the framework of European projects, these are based on Maturity Levels of the Software or service. The EOSC-hub guide proposes characteristics to help assess the maturity of a service via the operational definition of the Technology Readiness Level (TRL) indicators: TRL, 7, 8 and 9 [R35].

TRL 7 - Beta: "System prototype demonstration in operational environment".
TRL 8 - Production: "System complete and qualified"
TRL 9 - Production: "Actual system proven in operational environment"

CESSDA's Software Maturity Model (SMM) [R36], describes an approach for assessing the maturity of the components of the technical Research Infrastructure (RI), to meet as a prerequisite to supplying software artefacts for the RI. The Software Maturity grade is based on the Reuse Readiness Levels (RRLs), as developed by NASA Earth Science Data Systems. Each criteria is graded with 5 levels. The criteria are about: Documentation, Intellectual Property, Extensibility, Modularity, Packaging, Portability, Standards Compliance, Support, Verification and Testing, Security, Internationalisation and Localisation, Authentication and Authorisation.

2.2 Document management

The developed quality criteria are maintained in two documents: The "*Software Quality Assurance baseline document*" [R6], detailed in Section 3, and the "*Service Quality Assurance baseline document*" [R12], detailed in Section 4. These documents are managed as source code following principles described in [R6], namely:

- Are Version Controlled: GitHub repository as Version Control System (VCS). The versioning scheme follows the Semantic Version recommendation.
- Written in Markdown (MD) language, as such, any simple text editor can be used to read or write the documents.
- Open Access license Creative Commons Attribution-ShareAlike 4.0 International License [R13].
- Have a CONTRIBUTING.md in the source code repository specifying how a person can contribute to the document.
- Support: Corrections, additions and suggestions are handled with GitHub issues as issue tracker.
- Contributions to the document are handled through GitHub Pull Requests (PRs); where they are discussed, improved, approved and ending with a new release.
- Automatic build: a Jenkins CI job automaticallyThe developed quality criteria are maintained in t builds the document upon PR of the release to the master branch of the GitHub repository. It renders the MarkDown and produces the final document in PDF and HTML formats.

The final released documents are published in the DIGITAL.CSIC (https://digital.csic.es/) repository following the Open Access principles for research publications. The documents are also properly identified with Persistent IDentifiers (PIDs), in this case a Digital Object Identifier (DOI).

2.3 Metrics of the GitHub repositories

The Software *"Quality Assurance baseline document"* was initially established in the context of the INDIGO-DataCloud project and has three major releases. The most recent major release was issued by EOSC-Synergy and includes two minor releases. The Quality Assurance baseline for Services is a new development initiated by EOSC-Synergy and its first major release was issued in the Spring of 2020. The milestone M3.2 issued in April 2020 documents these releases.

Table 1 shows statistics regarding the Github repositories used to manage the quality criteria documents. The number of commits and first commit date do not reflect the real initial state of the repositories. The first commits are from an external repository used for the automatic build of the documents in PDF and HTML (manubot). This is reflected in the table and labeled “excluding manubot”, as such this second number (in bold), reflects the real number of commits for the document.

	Quality Assurance baseline	
Metric	Software	Service
Number of major releases	3	1
Number of minor releases	2	0
Number of commits / excluding manubot	331 / **84**	402 / **70**
Number of open issues	13	3
Number of closed issues	12	12
Number of Pull Requests	23	5
First commit date / excluding manubot	February 5th 2019	April 29th 2020

Table 1: Statistics of the two Github repositories; Software and Service Quality Assurance baseline documents.

2.4 Synergies with other projects

The Quality Assurance baseline documents [R6, R12] as well as the SQAaaS have been presented in several workshops and conferences. In particular, within the EOSC framework there are synergies towards applying these SQA models to other software components and services, external to the EOSC-Synergy project.

Furthermore, there have been meetings between EOSC-Synergy and CESSDA projects on how both projects can collaborate, and how CESSDA can contribute to the SQA criteria definitions. In this regard, the SQA baseline criteria and the CESSDA Software Maturity Model have been compared, and common points discussed.

Also on the scope of open data and open documentation, EOSC-Synergy has been collaborating with the FAIRsFAIR project on the area of FAIR data repository software and services with a focus on automation and quality assurance.

2.5 Use cases

The EOSC-Synergy WP4 is responsible for the development and integration of ten new high quality Thematic Services in EOSC. The quality criteria established by WP3 is being applied to these Thematic Services to assess and improve their quality. To this end, there is a tight cross work package collaboration in order to apply the Quality Assurance verification to the Software and Services, as described in milestone M4.1 (Inventory of Use Cases and EOSC Services). The ten Thematic Services are currently at different levels of implementing the Quality Assurance criteria.

The Worsica Thematic Service, in particular, has been selected due to its complexity as an early adopter of the Quality Assurance for purposes of testing the initial implementation and gathering of feedback. This approach enables the developers of the other Thematic Services to learn from this experience, while keeping the effort towards the validation of the SQA approach more focused. The Quality Assurance will be gradually applied to all Thematic Services.

Furthermore, this activity will have an invaluable contribution to improve the quality criteria and the technical developments, namely the CI/CD pipelines and the Software Quality Assurance as a Service (SQAaaS).

3. Software Quality Criteria

The *"Software Quality Assurance baseline document"* [R6] is managed through the GitHub repository: https://github.com/indigo-dc/sqa-baseline while the released documents are hosted at http://hdl.handle.net/10261/160086 with DOI: http://dx.doi.org/10.20350/digitalCSIC/12543.

EOSC-Synergy has already made (and published) the major release version v3.0, and two minor updates. As such, the latest version of the document as of June 2020 is v3.2.

3.1 Purpose and Goals

The *"Software Quality Assurance baseline document"* has been tailored upon the recommendations and requirements found in one of the first deliverables of the INDIGO-DataCloud project [R14]. These guidelines evolved throughout the project's lifetime and have been used in the DEEP-Hybrid-DataCloud and eXtreme DataCloud subsequent projects. Currently, EOSC-Synergy is the main supporter and manager of the document and its evolution.

The result is a consolidated Software Quality Assurance (SQA) baseline criteria emanated from the European Open Science Cloud (EOSC). It aims to outline the SQA principles to be considered in the software development efforts within the European research community, and continuously evolve in order to be aligned with current and future software engineering practices and security recommendations.

The goals of the document are the following:

1. Set the base of minimum SQA criteria that a software developed within an EOSC project SHOULD fulfill.
2. Enhance the visibility, accessibility and distribution of the produced source code through the alignment to the Open Source Definition [R16].
3. Promote code style standards to deliver good quality source code emphasizing its readability and reusability.
4. Improve the quality and reliability of software by covering different testing methods at development and pre-production stages.
5. Propose a change-based driven scenario where all new updates in the source code are continuously validated by the automated execution of the relevant tests.

6. Adopt an agile approach to effectively produce timely and audience-specific documentation.
7. Lower the barriers of software adoption by delivering quality documentation and the utilization of automated deployment solutions.
8. Encourage secure coding practices and security static analysis (SAST) at the development phase while providing recommendations on external security assessment.

3.2 Overview of the Software Quality Criteria

This section describes the quality conventions and best practices that apply to the development phase of a software component within the EOSC ecosystem. These guidelines ruled the software development process of the former successful European Commission-funded projects INDIGO-DataCloud, eXtreme DataCloud, and, DEEP-Hybrid-DataCloud where they have proved valuable for improving the reliability of software produced in the scientific European arena.

The SQA criteria aims at the development process driven by a change-based strategy, followed by a continuous integration approach. Changes in the source code trigger automated analysis of the new contributions in order to validate them before being added to the software component code base. Consequently, software components are more eligible for being deployed in production infrastructures, reducing the likelihood of service disruption.

Next, a summary of the criteria is shown. Each criterion has a codename to ease reference that is shown between square brackets:

1. **Code Accessibility [QC.Acc]**: Following the open-source model, the source code being produced MUST be open and publicly available to promote the adoption and augment the visibility of the software developments.
2. **Licensing [QC.Lic]**: As open-source software, source code MUST adhere to an open-source license [R39] to be freely used, modified and distributed by others.
3. **Code Workflow [QC.Wor]**: A change-based approach is accomplished with a branching model. Semantic Versioning specification [R17] is RECOMMENDED for tagging the production releases.
4. **Code Management [QC.Man]**: Recommendation for the existence of an issue tracking system, to track down both new enhancements and defects (bugs or documentation typos). Pull or Merge requests provide a place for

review and discussion of the changes proposed to be part of an existing version of the code.

5. **Code Style [QC.Sty]**: Code style requirements pursue the correct maintenance of the source code by the common agreement of a series of style conventions. These vary based on the programming language being used. Each individual software product MUST comply with community-driven or de-facto code style standards for the programming languages being used.
6. **Code metadata [QC.Met]**: Metadata for the software component provides a way to achieve its full identification, thus making software citation viable [R18]. It allows the assignment of a Digital Object Identifier (DOI) and is key towards preservation, discovery, reuse, and attribution of the software component. Thus, a metadata file SHOULD exist alongside the code, under its VCS.
7. **Unit Testing [QC.Uni]**: Unit testing evaluates all the possible flows in the internal design of the code, so that its behaviour becomes apparent. It is a key type of testing for early detection of failures in the development cycle. Minimum acceptable code coverage threshold SHOULD be 70%.
8. **Functional Testing [QC.Fun]**: Functional testing involves the verification of the software component's identified functionality, based on requested requirements and agreed design specifications. This type of software testing focuses on the evaluation of the functionality that the software component exposes, leaving apart any internal design analysis or side-effects to external systems. Functional testing SHOULD tend to cover the full set of functionality that the software component claims to provide.
9. **Integration Testing [QC.Int]**: Integration testing refers to the evaluation of the interactions among coupled software components or parts of a system that cooperate to achieve a given functionality. Integration testing outcome MUST guarantee the overall operation of the software component whenever new functionality is involved.
10. **Documentation [QC.Doc]**: MUST exist, be publicly available, how it should be managed and what types and formats should be used.
11. **Security [QC.Sec]**: Secure coding practices SHALL be applied into all the stages of a software component development lifecycle, such as: Compliance with Open Web Application Security Project (OWASP) secure coding guidelines [R18]. Perform static application security testing (SAST). Perform Dynamic application security testing (DAST).
12. **Code Review [QC.Rev]**: Code review MUST be done, it implies the informal, non-automated, peer, human-based revision of any change in the source code. It appears as the last step in the change management pipeline, once

the candidate change has successfully passed over the required set of change-based tests.

13. **Automated Deployment [QC.Aud]**: Production-ready code SHALL be deployed as a workable system with the minimal user or system administrator interaction leveraging software configuration management (SCM) tools.

4. Services Software Criteria

The *"Service Quality Assurance baseline document"* [R12] is managed through the GitHub repository: https://github.com/EOSC-synergy/service-qa-baseline while the released documents are hosted at https://digital.csic.es/handle/10261/214441 with the DOI: http://dx.doi.org/10.20350/digitalCSIC/12533.

EOSC-Synergy has already created and published the first major release of the Service Quality Assurance baseline identified with version "v1.0" and available on GitHub at https://github.com/EOSC-synergy/service-qa-baseline/releases/tag/v1.0 .

The Open Science realization in Europe is already taking its steps by means of the implementation of the European Open Science Cloud (EOSC). The EOSC aims at providing researchers with a unique, federated and inclusive view of fit-for-purpose services, developed and operated by the diverse European research infrastructures, including the underlying e-Infrastructures. Consequently, the ultimate success of the EOSC heavily relies on the quality aspects of those services, such as their stability or functional suitability.

The meaning of **Service** can be regarded from different perspectives. From an IT Service Management (ITSM) standpoint, such as the EOSC Service Management System (SMS) process model, a service is devised as a means to "provide value to the customer". The same goal is shared by the DevOps paradigm, but in this case there is a more pragmatic vision, the customer satisfaction is achieved through the continuous delivery of quality-assured services, with a shorter life cycle, as the final outcome of a comprehensive software development process.

The ITSM model has a broader focus. A service is an "intangible asset" that also includes additional activities such as customer engagement and support. Consequently, it is a much heavier process that might not be appropriate to be applicable for all types of services. The DevOps model, on the other hand, narrows down the scope to meet the user expectations by acting exclusively on the quality features of the service, which is seen as an aggregate of software components in operation.

4.1 Purpose and Goals

The baseline document [R12] provides an initial approach to Service Quality Assurance, meant to be applied in the integration process of services within the EOSC-Synergy project, which will be later accessible as part of the EOSC offerings.

The criteria compiled in the document favours a pragmatic and systematic approach that puts emphasis on the programmatic assessment of the quality conventions. To this end, the criteria compiled therein, builds on the DevOps culture already established in the preceding *"Software Quality Assurance baseline document"* [R6], to outline the set of good practices that seek the usability and reliability of services, and meet the user expectations in terms of functional requirements.

The proposed baseline criteria harnesses the capabilities of the quality factors in the underlying software to lay out the principles for attaining quality in the enabled services within the EOSC context. According to this view, service quality is the foundation to shape user-centric, reliable and fit-for-purpose services.

The Service Quality baseline aims at fulfilling the following goals by leveraging a DevOps approach:

- Complement with the existing approaches to assess and assure the quality and maturity of services within the EOSC, i.e. Technology Readiness Levels (TRLs) and EOSC Service Management System (SMS).
- Build trust on the users by strengthening the reliability and stability of the services, with a focus on the underlying software, thus ensuring a proper realization of the verification and validation processes.
- Ensure the functional suitability of the service by promoting testing techniques that check the compliance of the user requirements.
- Improve the usability by identifying the set of criteria that fosters the service adoption.
- Promote the automated validation of the service quality criteria.

4.2 Contextualization of a Service

As a result, a **Service**, as conceived in the baseline document, represents the following:

- **Web service** [R21]:

- A web service is an application or data source that is accessible via a standard web protocol (HTTP or HTTPS).
- Web services are designed to communicate with other programs, rather than directly with users.
- Most web services provide an API, or a set of functions and commands, that can be used to access the data.

- **Web application** [R22]:
  - A web application or "web app" is a software program that is delivered over the Internet and is accessed through a web browser.
- **Platform** or **Service Composition** [R23]:
  - Aggregation of multiple small services into larger services.
  - An integrated set of Web services, Web applications and software components.

Examples are: Web portals, Scientific portals and gateways, data services and repositories.

4.3 Overview of the Services Quality Criteria

This section describes the quality conventions and best practices that apply to the development, operation, and integration phases of a **Service** with a production infrastructure for research, such as the EOSC ecosystem. The guidelines rule the **Service** development and operation process within the framework of the EOSC-Synergy project.

Some of the criteria in this section is similar or based on the document "Software Quality Assurance baseline" [R6] and summarized in section 3.3:

1. **API Testing [SvcQC.Api]**: Web services commonly use application programming interfaces (APIs) to expose the available features to external consumers, which can be either oriented to the end-user or suitable for machine-to-machine communications. An accurate implementation of a publicly-accessible API is driven by a clearly defined specification. The OpenAPI Specification (OAS) [R24] provides the most suitable way to describe, compose, consume and validate APIs. The requirements assume the presence of such an API specification.
2. **Integration Testing [SvcQC.Int]**: Integration testing refers to the evaluation of the interactions among coupled **Services** or parts of a system that cooperate to achieve a given functionality.

3. **Functional tests [SvcQC.Fun]**: Functional testing is a type of black-box testing. It involves the verification of the **Services** identified functionality, based on requested requirements and agreed design specifications. This type of **Services** testing focuses on the evaluation of the functionality that the **Services** exposes, leaving apart any internal design analysis or side-effects to external systems.
4. **Performance tests [SvcQC.Per]**: Performance testing verifies that the service and the underlying software in execution, meets the specified performance requirements and assesses performance characteristics - for instance, capacity and response time [R25].
    a. **Stress or Load testing**, exercises the service and underlying software in execution, at the maximum design load, as well as beyond it, with the goal of determining the behavioral limits, and to test defense mechanisms in critical systems [R25]. *Stress testing is a subset of Performance testing* [R26].
    b. **Scalability testing** is a test methodology in which an application's or **Services** performance is measured in terms of its ability to scale up *and/or* scale out the number of user requests or other such performance measure attributes, through an increase in the amount of available resources. The definition is based on [R27]. *Scalability testing is a subset of Performance testing*.
    c. **Elasticity** is based on how quickly **Services** in an infrastructure are able to adapt [R27], in response to variable demand or workload for those service(s) [R28]. *Elasticity testing is a subset of Performance testing*.
5. **Documentation [SvcQC.Doc]**: Documentation is an integral part of any Software or Service delivery. For example, it describes how and what users can use and interact with it, or how operators can deploy, configure, and manage a given Software or Service.
6. **Security [SvcQC.Sec]**: Security assessment is essential for any production **Service**. While an effective implementation of the security requirements applies to every stage in the software development life cycle (SDLC), the security testing of a **Service** is also (similarly to the diverse testing strategies previously covered) a black-box type of testing. Hence, it focuses on the runtime analysis of security-related requirements, as part of the Dynamic Application Security Testing (DAST). Additionally, the compliance with security policies and regulations complements the analysis, which can be implemented, continuously validated and monitored through the Security as

Code (SaC) capabilities. SaC is a particularly suitable tool for endorsing security of **Service Composition** deployments.

7. **Policies [SvcQC.Pol]**: Policy documents describe what are the users expected behaviour when using the **Service**, how they can access it and what they can expect regarding privacy of their data. The documents are:
    a. Acceptable Usage Policy (AUP): Is a set of rules applied by the owner, creator or administrator of a network, website, or service, that restrict the ways in which the network, website or system may be used and sets guidelines as to how it should be used. The AUP can also be referred to as: Acceptable Use Policy or Fair Use Policy.
    b. Access Policy or Terms of Use: represent a binding legal contract between the users (and/or customers), and the Provider of the **Service**. The Access Policy mandates the users (and/or customers) access to and the use of the Provider's **Service**.
    c. Privacy Policy: Data privacy statement informing the users (and/or customers), about which personal data is collected and processed when they use and interact with the **Service**. It states which rights the users (and/or customers) have regarding the processing of their data.
8. **Support [SvcQC.Sup]**: Support is the formal way by which users and operators of the **Service** communicate with other operators and/or developers of the **Service**, in case of problems, be it operational problems or bugs in the **Service** or underlying Software. Reporting of enhancements, improvements and even documentation issues.
9. **Monitoring [SvcQC.Mon]**: Monitoring is a periodic testing of the **Service**. It requires a monitoring service from where tests are executed or sent and results of those tests are shown. The tests can be the same, in part or in total of the Functional tests. The technology used for the monitoring is left to the developers of the underlying software to decide eventually with input from the infrastructure(s), where the **Service** is foreseen to be integrated.
10. **Automated Deployment [SvcQC.Aud]**: The automated deployment of **Services** implies the use of code to install and configure them in the target infrastructures. Infrastructure as Code (IaC) templates allow operations teams to treat service provisioning and deployment in a similar fashion as developers manage the software code. Consequently, IaC enables the paradigm of immutable infrastructure deployment and maintenance, where **Services** are never updated, but deprovisioned and redeployed. An immutable infrastructure simplifies maintenance and enhances repeatability and reliability.

11. **Metrics [SvcQC.Met]**: A metric is a quantifiable measure that is used to track and assess the status of a specific process. In the case of **Services**, some relevant metrics are the number of users registered in the **Service**, or using it actively. Also accounting is important to track resource usage per user or group of users, either or both computing and storage resources. Although the metrics may be published in external services managed by the infrastructure, this is a common case in federated infrastructures such as EOSC.

5. Criteria Validation and Verification

The baseline criteria is aimed at a programmatic assessment of its quality conventions. In this regard the project is also developing quality assurance tools to facilitate the automated assessment of the quality criteria.

5.1 The SQA as a Service (SQAaaS)

The SQAaaS aims at facilitating the assessment of the quality of the research software by relying on the dynamic composition, execution, and analysis of the results obtained by the aforementioned CI/CD pipelines. The former software and service quality baselines provide the essential criteria to be considered, which are then implemented in the jenkins-pipeline-library [R19], and subsequently used in the pipelines.

One of the most prominent features of the SQAaaS is to be used as a software quality assessment tool. To this end, a comprehensive analysis will be offered to the end user by means of a i) report containing the output for each criterion identified in the baselines (only applicable to the criteria that can be checked programmatically), and the provision of a ii) quality badge (see Section 6) that recognizes the software quality achievements according to the EOSC-Synergy standards.

Consequently, the SQAaaS provides computational scientists with a tool to estimate the quality levels of their source code and software. The tool can be additionally used by other stakeholders in the research ecosystem, such as by the funders, to help in the decision making, or by the scientific users of the software, who are then able to ascertain the level of reliability and future sustainability of such software.

5.2 SQAaaS: initial steps

The SQAaaS architecture has been defined and its implementation is currently ongoing. The first prototype is planned for month 15 according to the project timeline and will be described in deliverable D3.2.

5.2.1 Main outcomes

As introduced before, the SQAaaS is a quality assessment tool --aka *Online Quality Assessment*-- as it inspects the source code to seek the fulfillment of the good practices identified in the quality baselines. It provides positive feedback about the quality achievements by means of quality badges, and identifies paths for improvement through the elaboration of an analysis report.

Consequently, the SQAaaS will foster the adoption of good practices in the software developed for research purposes. In this regard, an additional outcome --coined as *Pipeline*

as a Service-- will provide customized pipelines, composed through a graphical interface, for computational scientists that are interested in a subset of the capabilities provided by the tool. Table 2 summarizes the goals and expected results of the two aforementioned outcomes of the SQAaaS solution.

SQAaaS outcome	Goals	Expected output
Online Quality Assessment	Provide a comprehensive assessment of the quality of research software	• A report will be delivered containing the detailed information about the execution of each stage within the pipeline. • Quality badges will be issued to recognize the achievements according to the results obtained.
Pipeline as a Service	Lower the barriers/foster the adoption of quality practices in research software	• A ready-to-use Jenkins pipeline (Jenkinsfile) that is expected to be added to the user's software codebase. • Optionally, the user can choose to store (server-side) the pipeline in the Jenkins service provided by the EOSC-Synergy project.

Table 2: Goals and expected output of the main capabilities of the SQAaaS solution

5.2.2 Architecture

The high-level architecture is illustrated in Figure 2. The figure depicts the fundamental building blocks and how the above-introduced Online Quality Assessment and the Pipeline as a Service outcomes fit in the architecture.

Following a bottom-up analysis, the SQAaaS backend is composed of three components, which are in charge of implementing the workflows, complemented by the *SQAaaS API*, that manages the incoming requests and triggers the appropriate workflow. The internal components are:

- The *Pipeline Composer* leverages the Pipeline as Code capabilities from Jenkins framework [R19] to create on-demand pipelines ready to be executed by Jenkins CI system. As depicted in the figure, both SQAaaS outcomes rely on this component, either to obtain a pipeline matching the SQA baseline criteria --needed by the *Online Quality Assessment*--, or to compose a customized pipeline from the capabilities offered by the jenkins-pipeline-library according to the user input --as needed by the *Pipeline-as-a-Service* outcome--.
- The *Jenkins CI system* carries out the execution of the pipelines, managing the provisioning of resources, and displays the results obtained.
- The *Badge Issuing system* corresponds to the framework that provides the issuing of digital badges according to the results from the pipeline execution. In the SQAaaS

architecture, only the Online Quality Assessment outcome triggers this component, in order to reflect the adherence of the analyzed software with the criteria defined in the software and service baselines. Next section thoroughly covers the discussion of the badge-related topics.

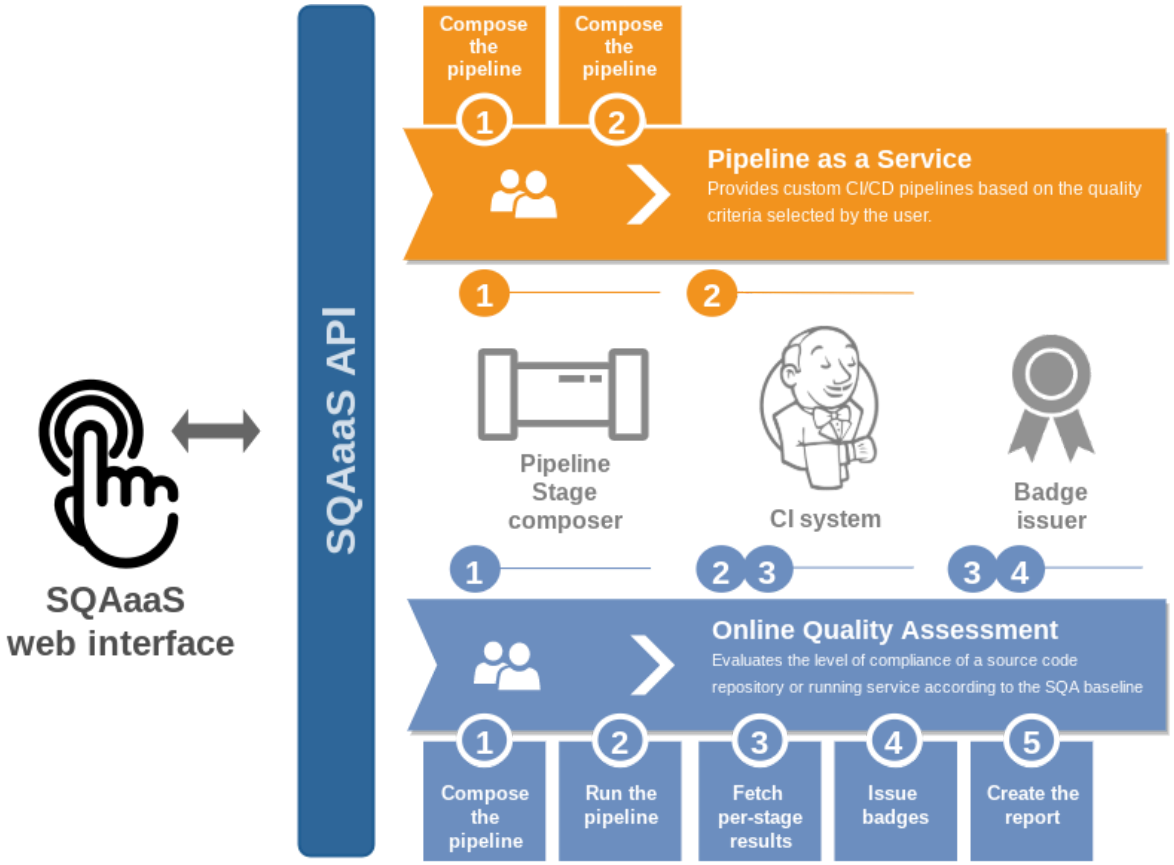

Figure 2: SQAaaS high-level architecture

Last but not least, the SQAaaS frontend consists of a web interface that gathers the user selections and makes the request to the appropriate API endpoint that will eventually trigger the required outcome. The frontend will provide --and therefore the API-- extended capabilities to display the reports with the results from the Online Quality Assessment, store and retrieve them, as well as the ability to test the execution of pipelines in the case of the Pipeline-as-a-Service outcome.

6. Quality Badges

Traditional physical badges have been used as a way to prove membership and to demonstrate attaining a certain achievement. Traditional badges manifest as identity cards, attendance certificates, certificates of achievement, etc. However, this type of certificates cannot satisfy the requirements for online sharing, verification, portability and the inability to be tampered.

Instead, digital badges represent virtual certificates that can easily be shared, watched and verified online [R2]. As is the case of physical badges, a digital badge still represents an achievement attained by a certain entity and demonstrates a quality seal for others to compare different entities on the basis of the quality level determined by the achieved badge.

6.1 Purpose

The proper recognition for software compliant with the quality levels defined in the EOSC-Synergy project requires issuing quality badges that can be automatically verified, can't be tampered, and represent the achievement made by the software or service in the quest for quality. To this aim, this section focuses on the adoption of quality badges that visually express the adherence of the software to the established quality criteria levels defined within the project and which were described in section 3. Notice that these digital badges can also be used to certify the quality of the service according to the quality metrics defined in section 4.

6.2 State of the Art

As part of the initial technology scouting carried out in the project regarding digital badge issuing, a comprehensive whitepaper titled "State of the Art Regarding Digital Badge Issuing Technology" [R1] was produced. The paper analyses the existing services and tools that allow issuing digital badges with the purpose of selecting the most appropriate for adoption by EOSC-Synergy. The badges will be used to issue proper recognition stamps to software that complies with the Software Quality Levels (SQA) metrics defined in the project.

This subsection provides a brief summary of the contents of this document for the sake of completeness. For further details, the reader is instructed to read the aforementioned whitepaper [R1].

6.2.1 Open Badges: Specification for Digital Badges

Open Badges is the leading standard for digital credentials originally developed by Mozilla and now managed by IMS Global Learning Consortium® (http://www.imsglobal.org/). These are based on the Open Badges specification [R7], described as JSON-LD context

(v2.0). The badges contain detailed metadata (depicted in Figure 3) about achievements. Indeed, an Open Badge is a PNG or SVG file that has been by leveraging a DevOps approach modified to adhere to the Open Badges Baking Specification [R37] which includes an iTxt section in the PNG file or some markup section inside the SVG file and can include a JSON Web Signature (JWS) for a http://www.imsglobal.org/signed assertion. To issue Open Badges you need a technology platform that supports the Open Badges Specification and there are different Web/Cloud-based products that support Open Badges v2.0 as indicated in the IMS product certifications list [R38].

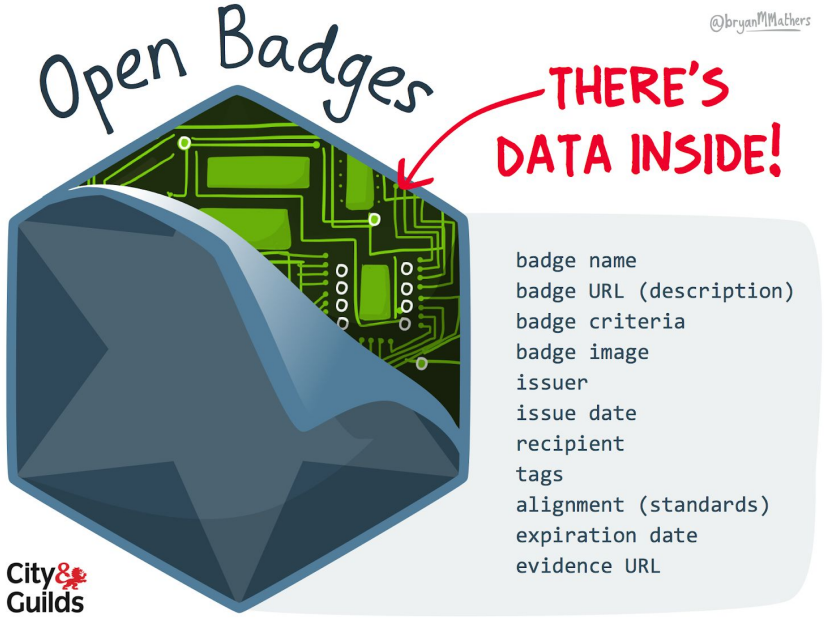

Figure 3: Open Badges: A digital badge. Source: [R3]

As indicated in [R8], issuing Open Badges requires constructing and publishing a set of interconnected resources that follow the structure and guidelines set out in the Open Badges Specification [R7]. For each badge awarded, there's:

- An Issuer Profile describing the individual or organization awarding badges. The information in the profile will appear in the metadata for all badges, including name, description, contact email address, and website URI. One Issuer profile is typically shared between all the badges that an organization awards.
- A BadgeClass, the formal description of a single achievement the Issuer recognizes. This includes information such as the name, description, and of course the graphic image that's the visual face of the badge, but also links to detailed criteria for how the badge may be earned and the Issuer profile that created it. A human readable criteria

page and an image file visually symbolizing the accomplishment must be published at a stable URLs

- An Assertion, the record of an individual's achievement of the badge. The Assertion links to one BadgeClass and contains the information specific to one Recipient's achievement of the badge's criteria, like the date it was awarded, the encoded Recipient identifier it was awarded to, and optionally a link to evidence and an expiration date. An Assertion is the entry point for badge verification, and it may be delivered either as a hosted object with an accessible URL alongside the BadgeClass and Issuer profile resources, or as a cryptographically signed document given to the Recipient in order to distribute to relevant Consumers. A single BadgeClass may be awarded to many different individuals by creating an Assertion for each Recipient.

Badge Assertions may be revoked if the Issuer determines, they were issued in error or no longer should be valid. For a hosted Assertion, revocation entails replacing the Assertion with a note declaring the reason for revocation. For signed Assertions, Issuers may create a revocation list with keys for each of the UIDs they wish to revoke. A revocation list is a single JSON document for the entire issuer.

The badge issuing flow is:

- The issuer has already defined and hosted an Issuer Profile and at least one BadgeClass.
- The issuer creates and hosts (or cryptographically signs) a badge Assertion.
- The issuer delivers the badge to the Recipient, typically as a baked image file.

Any individual or organization can create an Issuer profile and begin defining and issuing Open Badges. Any entity that can be described with a name, description, URL, image, and email address is a possible candidate to become an Issuer. To issue Open Badges you need a technology platform that supports the Open Badges Specification [16], using either a Cloud-service to support issuing badges or creating a new Open Badges issuer application, as described in the Open Badges v2.0 specification. The most appropriate tool for the later, is Badgr [R9], though there also exist other alternatives which are covered in the following sections.

6.2.2 Open-Source Tools to Issue Open Badges

The Whitepaper produced by EOSC-Synergy [R1], contains a list of Managed Services compliant with the Open Badges specification as well as a list of open-source tools with the same purpose. Table 3 summarises those open-source tools.

Name	Logo	Motto
Badgr - https://badgr.com/	badgr	Make your badges meaningful with Badgr The world's leading open source digital badge platform
BlockCerts - https://www.blockcerts.org	BLOCKCERTS	The Open Standard for Blockchain Credentials

Table 3: Open-source tools compliant with the Open Badges specification.

EOSC-Synergy has adopted Badgr as the underlying platform to issue digital badges for the following reasons:

- Is an open-source development (coded in Python with Apache 2.0 license) and contributions are welcome by the product owners (Concentric Sky, a software development company).
- Has a fully documented API that can be used in order to programatically issue the Badges without any human intervention.
- Offers a web-based multi-tenant Graphical User Interface that can be customized and rebranded to fit the project's needs.
- Offers different user roles (administrator vs user) in order to gain access to advanced functionality of the platform.
- Allows to define additional Badge designs to be used for several purposes.
- Offers OAuth2 Identity Provider functionality to help connected apps to securely obtain a user-specific API token to use to access the user's badges.

6.3 Badgr Deployment in EOSC-Synergy

In order to facilitate the deployment and operation of Badgr in EOSC-Synergy, a set of Dockerfiles were created to deploy the main components: Badgr Server and Badgr UI and are publicly available in the corresponding GitHub repository [R4]. Figure AII-1 in Annex II shows a Dockerfile to deploy the Badgr service.

Encapsulating all the dependencies into an Ubuntu 18.04 Docker image facilitates the ability to deploy instances of the Badgr Server. Also, this facilitates running a highly-available instance inside a Container Orchestration Platform (COP) such as Kubernetes.

The user interface has also been encapsulated as a Docker container image in order to facilitate its deployment, as shown in the Figure AII-2 in Annex II.

A container image based on Node.JS is employed and logos from EOSC-Synergy are included in order to have proper branding of the deployed instance.

Since both Badgr Server and Badgr UI should run simultaneously in order to use its functionality, Docker Compose has been used as the tool to facilitate its coordinated deployment. For this, the docker-compose.yml file depicted in Figure AII-3 in Annex II has been created.

An instance of Badgr has been deployed for EOSC-Synergy available in [R5] for issuing badges. For the time being, this instance is used for testing and development purposes. A production instance will be made available with the second prototype of the SQAaaS.

6.4 Badges Graphic Design

In order to choose the graphic design of the badges to be adopted by EOSC-Synergy, several graphical designs were produced. A subset of designs is shown in Figure AIII-1 in Annex III as well as the full list of designs shown in Figure AIII-2.

A poll was run through March 2020 to all the project participants, which could cast a vote for up to their three most preferred designs. 20 people from 12 institutions (UPV, CSIC, LIP, EGI, CNB, IISAS, CIEMAT, INCD, KIT, FCT, PSNC and DANS) responded to the call.

The winning design was slightly modified in order to read “EOSC-SYNERGY compliant” instead of “EOSC compliant” for the sake of inclusiveness. The final badge was also adapted to have a three-level classification (silver, bronze and gold) as depicted in Figure 4.

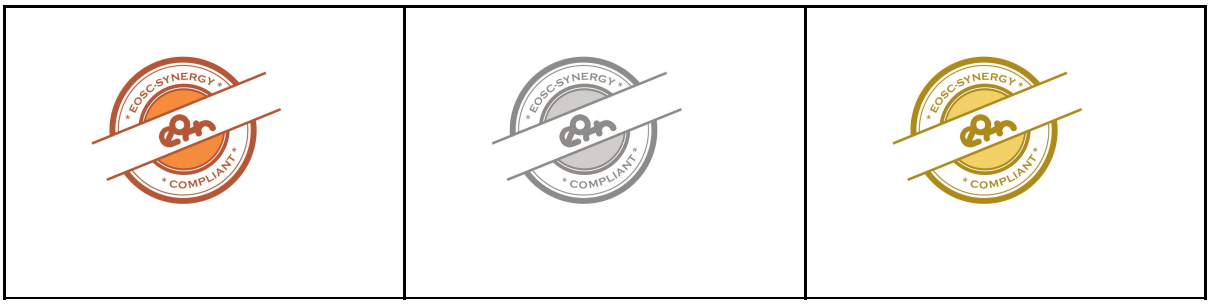

Figure 4: Final design with the three-level classifications.

For the sake of completeness, the process of issuing badges with Badger, both using the GUI and the API is summarised in Annex I.

7 Conclusions

EOSC-Synergy has defined two sets of baseline criteria covering a wide range of quality aspects concerning software development as well as service deployment and delivery. The proposed baseline criteria, harnesses the capabilities of the quality factors in the underlying software to lay out the principles for attaining quality in the enabled services within the EOSC context.

With a strong focus on automation and programmatic conformance assessment, the quality criteria constitutes the foundation of the EOSC-Synergy software quality assurance as a service (SQAaaS). The SQAaaS aims at facilitating the on-demand automated assessment of the quality by relying on the dynamic composition, execution, and analysis of the quality assessment performed through CI/CD pipelines.

The final step of the SQAaaS will be the issuance of quality badges asserting the level of compliance towards the baseline criteria. The badges will be based on the Open Badges specification, and will be assigned to URLs associated to software repositories or service instances. The badges will provide a powerful visual assertion of quality that will contribute to establish trust, promote adoption and reward the efforts of the software developers and service providers towards quality.

8 References

R1	Moltó, Germán; Campos, Isabel ; Hardt, Marcus; Blanquer, Ignacio ; Caballer, Miguel; Orviz, Pablo ; David, Mario; Gomes, Jorge. "State of the Art Regarding Digital Badge Issuing Technologies". http://dx.doi.org/10.20350/digitalCSIC/12505
R2	MY OPEN BADGE - Digital Badges: https://myopenbadge.com/en/open-badge-en/digital-badges/
R3	Issuing Open Badges. https://openbadges.org/get-started/issuing-badges/
R4	Badgr Dockerfiles. https://github.com/EOSC-synergy/badgr
R5	EOSC-Synergy's Badgr Instance: https://badges.eosc-synergy.eu/
R6	Pablo Orviz, Alvaro Lopez, Doina Cristina Duma, Mario David, Jorge Gomes, Giacinto Donvito, "A set of Common Software Quality Assurance Baseline Criteria for Research Projects", 2017, http://dx.doi.org/10.20350/digitalCSIC/12543 (Github repository: https://github.com/indigo-dc/sqa-baseline)
R7	Open Badges v2.0 IMS Final Release. https://www.imsglobal.org/sites/default/files/Badges/OBv2p0Final/index.html
R8	OpenBadges. Developers Guide. https://openbadges.org/developers/
R9	Badgr. https://info.badgr.com/
R10	Badgr App Developers API Guide. https://badgr.org/app-developers/api-guide/
R11	Postman. https://www.postman.com/
R12	Orviz Fernández, Pablo ; Mario David; Jorge Gomes; Joao Pina; Samuel Bernardo; Campos Plasencia, Isabel ; Germán Moltó; Miguel Caballer, "EOSC-Synergy: A set of Common Service Quality Assurance Baseline Criteria for Research Projects", June 2020: DOI http://dx.doi.org/10.20350/digitalCSIC/12533 (Github repository: https://github.com/EOSC-synergy/service-qa-baseline)
R13	Creative Commons, Open Access licenses: https://creativecommons.org/about/program-areas/open-access/
R14	Jorge Gomes, Mário David, Cristina Aiftimiei, Pablo Orviz, Peter Solagna, Elisabetta Ronchieri, Ian Nielsen; Initial Plan for Software Management and Pilot Services: https://owncloud.indigo-datacloud.eu/index.php/s/yDklCrWjKnjutVA
R15	Scott Bradner; Key words for use in RFCs to Indicate Requirement Levels, (1997): https://www.ietf.org/rfc/rfc2119.txt
R16	The Open Source Definition: https://opensource.org/osd
R17	Semantic Versioning specification: https://semver.org
R18	Software citation principles:http://dx.doi.org/10.7717/peerj-cs.86
R19	Pablo Orviz, (2020), An Automation-Driven Quality Assurance strategy for Research Software as the vehicle to consolidate the European Open Science realisation, (Unpublished doctoral dissertation)

R20	Pipeline as Code in Jenkins: https://www.jenkins.io/solutions/pipeline/
R21	Web service: https://techterms.com/definition/web_service
R22	Web application: https://techterms.com/definition/web_application
R23	Platform or Service Composition: https://csrc.nist.gov/glossary/term/Service_Composition
R24	The OpenAPI Specification (OAS): https://www.openapis.org/
R25	Performance testing: http://www.swebok.org
R26	Performance and Stress testing: https://www.geeksforgeeks.org/difference-between-performance-and-stress-testing/
R27	Scalability, Elasticity, and Efficiency in Cloud Computing: a Systematic Literature Review of Definitions and Metrics, Sebastian Lehrig, Hendrik Eikerling, Ste en Becker, Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures (2015-05-04) https://doi.org/10.1145/2737182.2737185 , DOI: 10.1145/2737182.2737185 · ISBN: 9781450334709
R28	Scalability analysis comparisons of cloud-based software services, Amro Al-Said Ahmad, Peter Andras, Journal of Cloud Computing (2019-07-23), DOI: https://doi.org/10.1186/s13677-019-0134-y
R29	ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models: https://www.iso.org/standard/35733.html
R30	Estdale, John & Georgiadou, Elli. (2018). Applying the ISO/IEC 25010 Quality Models to Software Product: 25th European Conference, EuroSPI 2018, Bilbao, Spain, September 5-7, 2018, Proceedings. DOI: https://doi.org/10.1007/978-3-319-97925-0_42.
R31	Galli, Tamas, Francisco Chiclana, and Francois Siewe. "Software Product Quality Models, Developments, Trends and Evaluation." SN Computer Science, 1:154, (2020).
R32	Adewumi, Adewole, Sanjay Misra, and Nicholas Omoregbe. "A review of models for evaluating quality in open source software." 2013 International Conference on Electronic Engineering and Computer Science, IERI Procedia 4 (2013): 88-92.
R33	Humble, J. and Farley, D. (2011), Continuous delivery: Reliable software releases through build, test, and deployment automation, A Martin Fowler signature book, Addison-Wesley, Upper Saddle River.
R34	König, Leon, and Andreas Steffens. "Towards a quality model for devops." Continuous Software Engineering & Full-scale Software Engineering (2018): 37.
R35	EOSC-hub Service Maturity Classification: https://wiki.eosc-hub.eu/display/EOSC/Service+Maturity+Classification
R36	John Shepherdson, CESSDA Software Maturity Levels (2019), DOI: 10.5281/zenodo.2591055
R37	Open Badges Baking specification. https://www.imsglobal.org/sites/default/files/Badges/OBv2p0Final/baking/index.html
R38	IMS GLOBAL product certifications: https://site.imsglobal.org/certifications
R39	Licenses & Standards, Open Source Initiative: https://opensource.org/licenses

9 Annex I - Badges Issuance with Badgr

Badgr is flexible enough to accommodate different types of users. One can use the web-based Graphical User Interface (GUI) to streamline the process of issuing the badges. However, EOSC-Synergy is more interested in leveraging Badgr's API to provide programmatic issuing of badges. This annex briefly describes both approaches.

9.1 Issuing Badges through the GUI

Figure AI-1 shows a screenshot of the interface including an issuer, the component responsible to issue the badge, and three badges created for testing purposes.

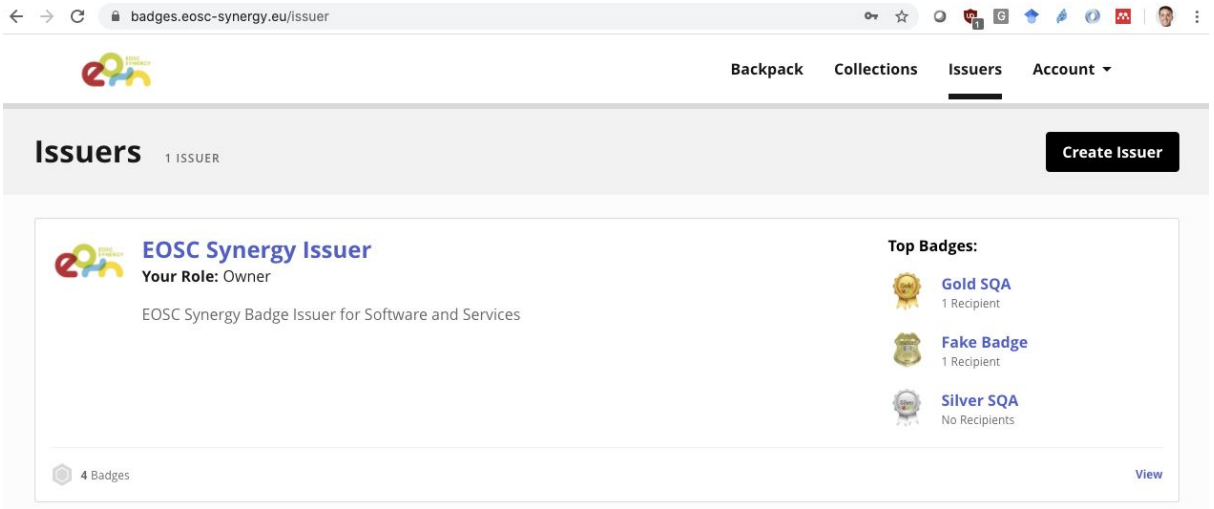

Figure AI-1: EOSC Synergy Issuer populated with sample badges for testing purposes.

Badgr provides an end-to-end solution to create the badges, award them to recipients and verify them within the platform. Figure AI-2 shows a list of awarded badges.

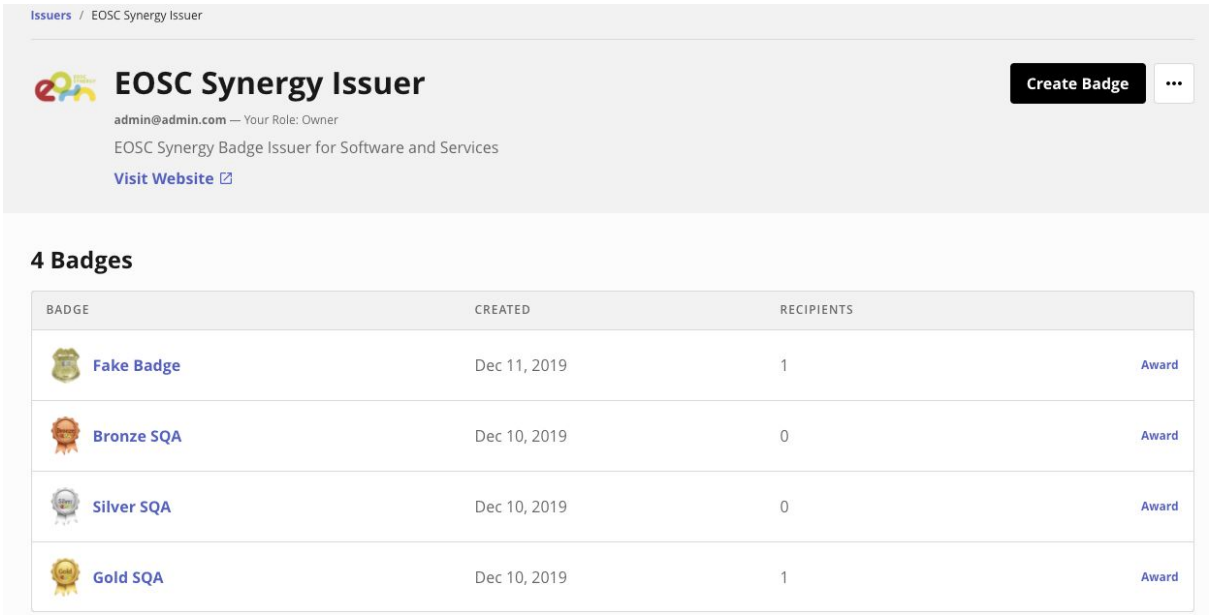

Figure AI-2: A sample list of badges awarded by the issuer, for testing purposes.

Badges can be awarded to recipients which can be identified via an e-mail account or URL. This allows awarding a badge to a given version of a software uniquely identified by an URL or to an external distributed source code management system such as GitHub, as shown in Figure AI-3.

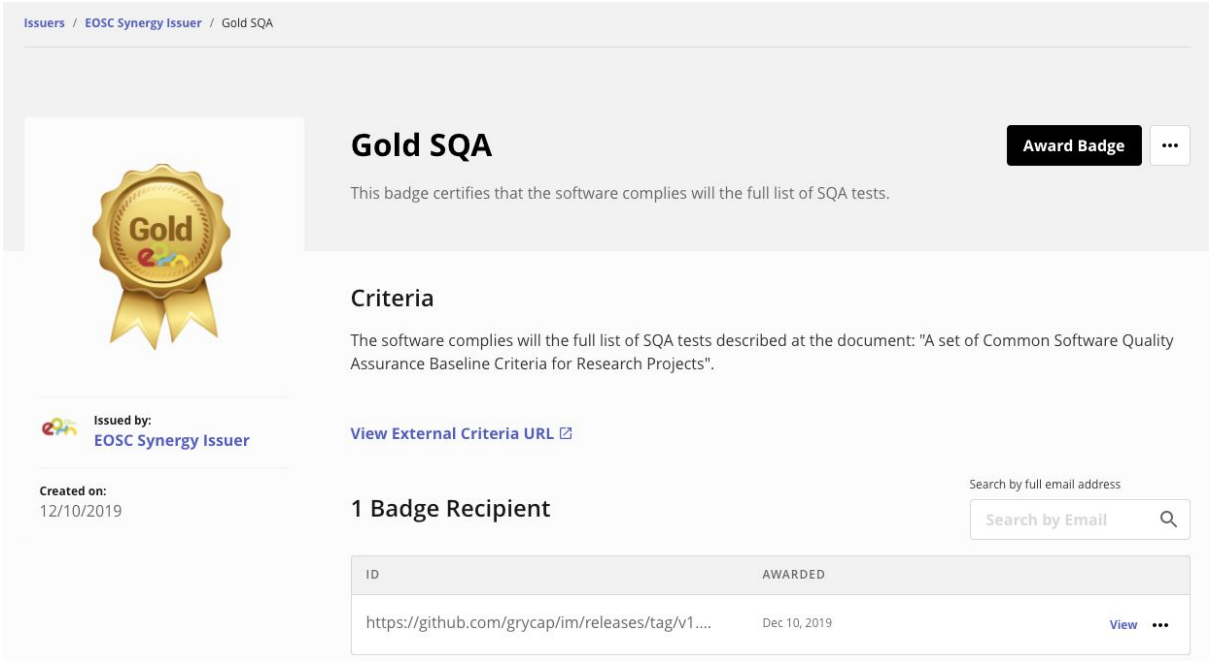

Figure AI-3: A list of badges awarded. Notice that the recipient identifies a specific version of a software, located by an URL.

Each badge includes information about:

- The criteria satisfied by the software in order to have deserved receiving such a badge. This can be identified as an external URL that provides further information. For the case of software quality, we rely on the document *"A Set of Common Software Quality Assurance Baseline Criteria for Research Projects"* [R6].
- The evidence, which can be an external URL that provides a report of the achievements of the recipient to have deserved the award.

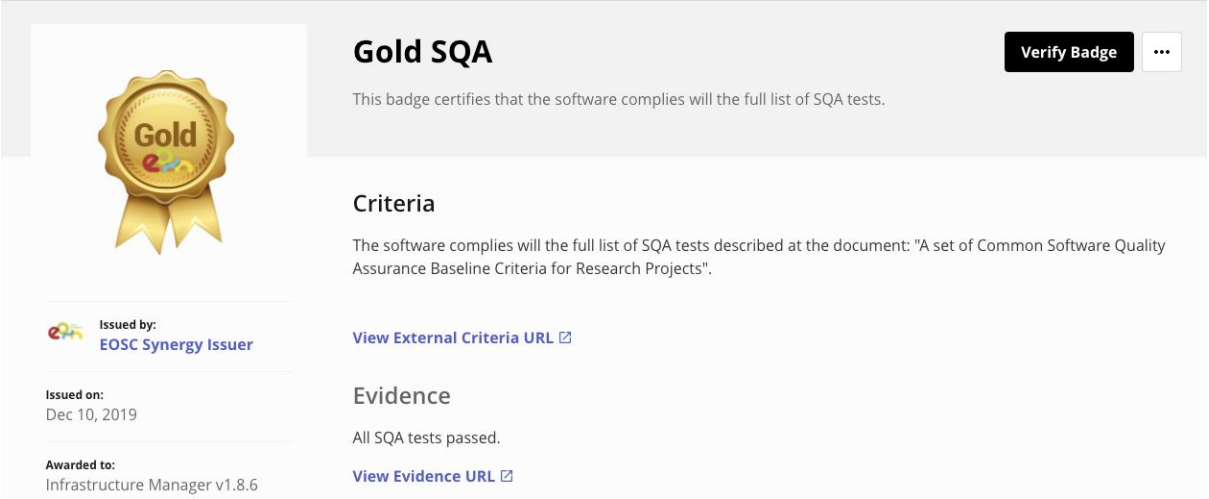

Figure AI-4: A sample badge awarded to a specific version of a software.

In the case of software, this can point to a publicly available Continuous Integration (CI) system that automatically includes the relevant information about the compliance of the software with certain quality metrics (style analysis, unit testing coverage, functional testing, etc.) as shown in the following Figure AI-5.

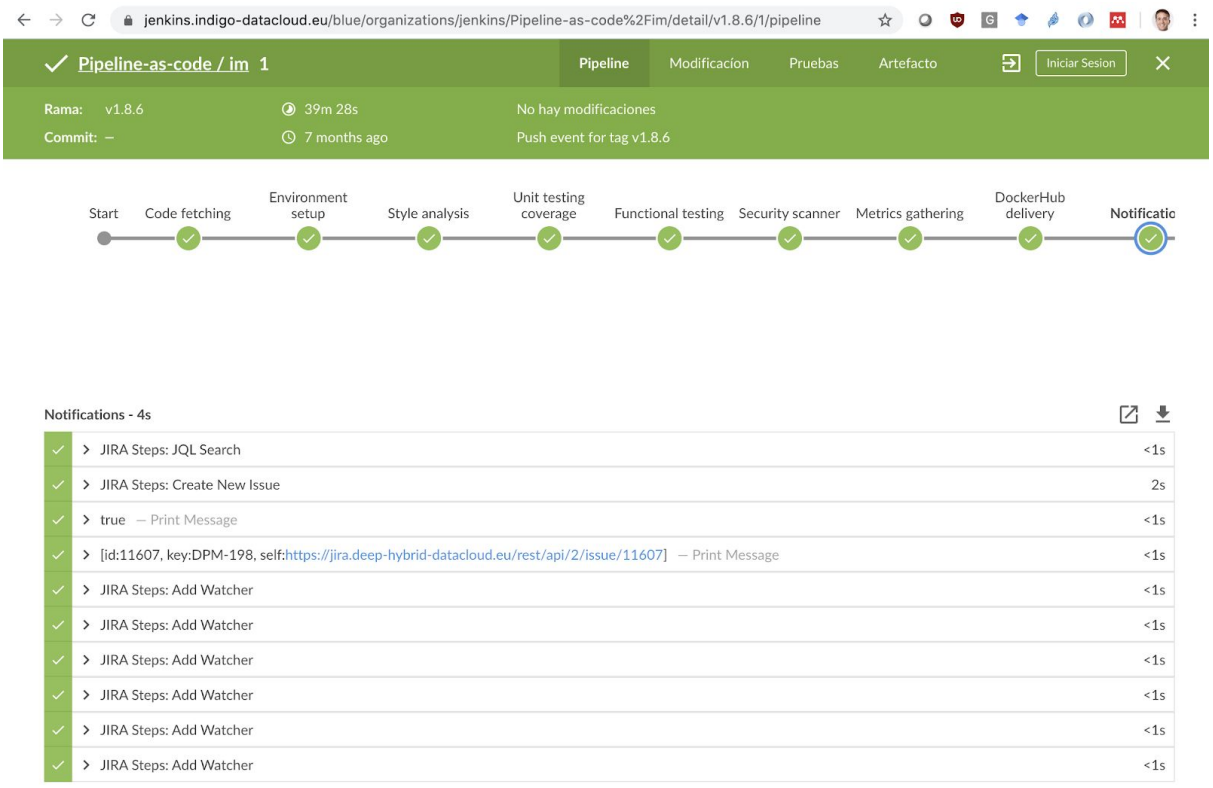

Figure AI-5: a sample report obtained by Jenkins concerning a certain version of a software, to be used as evidence for the awarded badge.

9.2 Issuing Badges through the API

Badgr provides a fully-featured API, documented in [R10] that provides programmatic access to functionality such as authentication, creating an Issuer, defining a BadgeClass, and issuing an Assertion. Badgr uses OAuth2 for most operations. As an API client user, you can obtain an OAuth2 Bearer Token on behalf of your own Badgr user account using a password-based grant.

Since the Badgr API is extensive, this section just provides an overview of the functionality strictly related to the project's goals. We will be using Postman [R11], the collaboration platform for API development to exemplify the API calls to be done and also curl, for the sake of diversity.

Obtaining a List of BadgeClasses

One can obtain a list of BadgeClasses (the badges that can be awarded) already created in the Badgr instance by issuing the following command:

```
curl --location --request GET
'https://badges.eosc-synergy.eu:8443/v2/badgeclasses' \
--header 'Content-Type: application/json' \
```

```
--header 'Authorization: Bearer rH54LQ5wjRCi04rwPOQDG5JdvG9AJP' \
--data-raw ''
```

Issuing an Assertion

This will be the most used operation since it involves awarding a certain Badge to a certain software or service. Figure AI-6 shows an example POST request to the Badgr instance in order to issue an assertion for a sample project. You will notice that the Body of the request includes:

- The recipient
  - This indicates who will be awarded the badge. In this example case, it is a certain software release identified by an URL
- The narrative
  - This is a text-based human-readable message to provide further hints on what this badge assertion is about.
- The evidence
  - This includes a URL that can point to a report with the detailed evaluation of the achievements carried out by the recipient (in this case the software version) to receive such a badge.
- The expiration date (optional)
  - This field can be used to allow recipients to go under a periodic recertification process to achieve quality assessment on a regular basis.

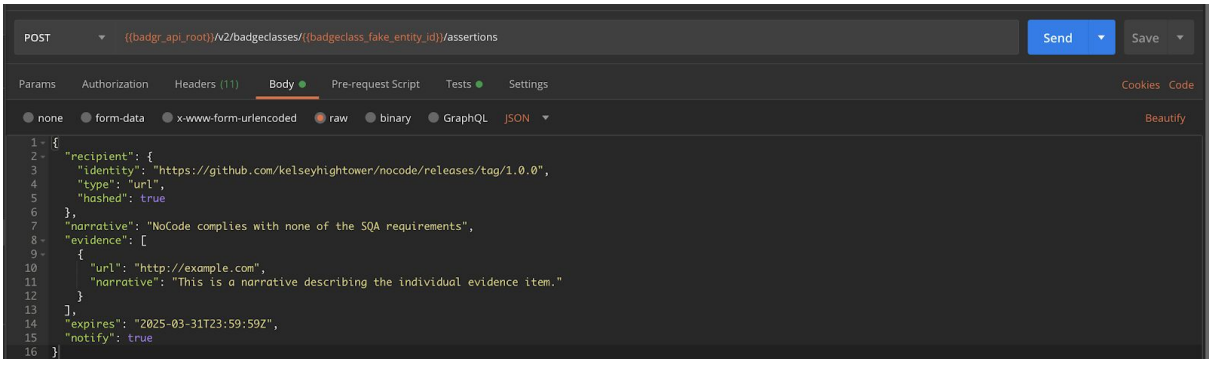

Figure AI-6: Screenshot of Postman to make a POST request to issue a Badge, e.g. create an Assertion, whose BadgeClass description is shown in Figure AIII-7.

```
{
    "status": {
```

```
        "description": "ok",
        "success": true
    },
    "result": [
        {
            "entityType": "BadgeClass",
            "entityId": "DmvZjWC6SouLTiy6psBJlg",
            "openBadgeId":
"https://badges.eosc-synergy.eu:8443/public/badges/DmvZjWC6SouLTiy6psBJlg",
            "createdAt": "2019-12-10T11:21:07.935391Z",
            "createdBy": "iG8vNffcT-Ccau9VsgkqNw",
            "issuer": "05ARn1mBRqOUbBJRgdOFew",
            "issuerOpenBadgeId":
"https://badges.eosc-synergy.eu:8443/public/issuers/05ARn1mBRqOUbBJRgdOFew",
            "name": "Gold SQA",
            "image":
"https://badges.eosc-synergy.eu:8443/media/uploads/badges/issuer_badgeclass_4364
3075-54df-4487-92ff-f962e252e02f.png",
            "description": "This badge certifies that the software complies will
the full list of SQA tests.",
            "criteriaUrl":
"https://indigo-dc.github.io/sqa-baseline/v/f835d0a5f8369b92072f3d05939078da10f6
f7bd/",
            "criteriaNarrative": "The software complies will the full list of
SQA tests described at the document: \"A set of Common Software Quality
Assurance Baseline Criteria for Research Projects\".",
            "alignments": [],
            "tags": [],
            "expires": {
                "amount": null,
                "duration": null
            },
            "extensions": {}
        },
...
```


Figure AI-7: Excerpt of the JSON output for a BadgeClass.

10 Annex II - Docker related files for Badgr

```
FROM ubuntu:18.04
LABEL maintainer="Miguel Caballer <micafer1@upv.es>"
LABEL version="v2.24.1-1"
LABEL description="Badgr server image"
EXPOSE 8800

RUN apt-get update && apt-get install --no-install-recommends -y \
    apache2 \
    libapache2-mod-wsgi \
    git \
    git-core  \
    gcc  \
    python-pip \
    python-dev \
    libjpeg-turbo8 \
    libjpeg-turbo8-dev \
    swig \
    libxslt-dev \
    automake \
    autoconf \
    libtool \
    libffi-dev \
    libssl-dev \
    libmysqlclient-dev \
    python-cairo \
    python-setuptools && \
    mkdir badgr && cd badgr && \
    git clone https://github.com/concentricsky/badgr-server.git --branch=v2.24.1
code  && \
    pip install wheel   && \
    pip install -r /badgr/code/requirements.txt && \
    pip install /badgr/code && \
    apt-get purge -y gcc git python-dev python-pip libmysqld-dev libssl-dev
libxslt-dev libjpeg-turbo8-dev automake autoconf git-core git && \
    apt-get autoremove -y && apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/*
/var/tmp/* && rm -rf ~/.cache/

# Enable SSL
RUN ln -s /etc/apache2/mods-available/ssl.load /etc/apache2/mods-enabled/ssl.load
# Copy dummy certificates
COPY server.crt /etc/ssl/certs/server.crt
COPY server.key /etc/ssl/certs/server.key

# Change ports
RUN sed -i -e 's/Listen 80/Listen 8000/g' /etc/apache2/ports.conf
RUN sed -i -e 's/Listen 443/Listen 8443/g' /etc/apache2/ports.conf

# Copy badgr app configuration
COPY badgr.conf /etc/apache2/conf-enabled/badgr.conf
# Create log dir
RUN mkdir /badgr/code/logs

# Copy app local files
COPY settings_local.py /badgr/code/apps/mainsite/settings_local.py
COPY wsgi.py /badgr/code/wsgi.py
COPY local.sqlite3 /badgr/code/local.sqlite3
```

```
# Set correct perms
RUN chown www-data -R /badgr/code

CMD ["/usr/sbin/apache2ctl", "-DFOREGROUND"]
```

Figure AII-1: Dockerfile to deploy Badgr Server.

```
FROM node:8.16.2-jessie as node

RUN git clone https://github.com/concentricsky/badgr-ui.git /badgr-ui
--branch=v2.24.45 && \
       cd /badgr-ui

COPY environment.prod.ts /badgr-ui/src/environments

# Patch to fix sh error with test
RUN  sed -i '/"verify-prod-environment":/c\
"verify-prod-environment": "/bin/true",' /badgr-ui/package.json

WORKDIR /badgr-ui

RUN    npm install

COPY os-logo-large.svg
/badgr-ui/node_modules/@concentricsky/badgr-style/dist/images/os-logo-large.svg
COPY os-logo-small.svg
/badgr-ui/node_modules/@concentricsky/badgr-style/dist/images/os-logo-small.svg

RUN    npm run build

RUN ls -l /badgr-ui

# Stage 2
FROM nginx:stable-alpine
LABEL maintainer="Miguel Caballer <micafer1@upv.es>"
LABEL version="v2.24.45-2"
LABEL description="Badgr UI image"

COPY --from=node /badgr-ui/dist /usr/share/nginx/html

COPY nginx.conf /etc/nginx/nginx.conf

ENV BADGRSERVER http://badgrserver:8000

COPY run.sh /run.sh

RUN chmod 755 /run.sh

CMD ["/run.sh"]
```

Figure AII-2: Dockerfile to deploy Badgr UI.

```
version: '3.3'

services:
   db:
     image: mysql:5.7
     volumes:
       - /tmp/mysql:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: OMMITTED
       MYSQL_DATABASE: OMMITTED
       MYSQL_USER: OMMITTED
       MYSQL_PASSWORD: OMMITTED

   badgrserver:
     depends_on:
       - db
     image: eoscsynergy/badgr-server
     ports:
       - "8000:8000"
       - "8443:8443"
     volumes:
       - /tmp/badgr/settings_local.py:/badgr/code/apps/mainsite/settings_local.py
       - /tmp/badgr/server.crt:/etc/ssl/certs/server.crt
       - /tmp/badgr/server.key:/etc/ssl/certs/server.key
       - /tmp/badgr/mediafiles:/badgr/code/mediafiles
     restart: always

   badgrui:
     depends_on:
       - badgrserver
     image: eoscsynergy/badgr-ui
     ports:
       - "80:80"
       - "443:443"
     restart: always
     volumes:
       - /tmp/badgr/nginx.conf:/etc/nginx/nginx.conf
       - /tmp/badgr/server.crt:/etc/ssl/certs/server.crt
       - /tmp/badgr/server.key:/etc/ssl/certs/server.key
     environment:
       BADGRSERVER: "https://localhost:8443"

volumes:
    db_data: {}
```

Figure AII-3: Docker Compose file to perform a coordinated deployment of a Badgr instance.

11 Annex III - List of Designed Badges

This Annex includes, for the sake of completeness, both the subset and the complete set of badges that were designed as part of the decision process to choose the final design for the badges.

Figure AIII-1: A subset of the designed digital badges to be adopted by the project.

B11

B12

B13

B14

B23

B24

B31

B32

B41

B42

B43

B44

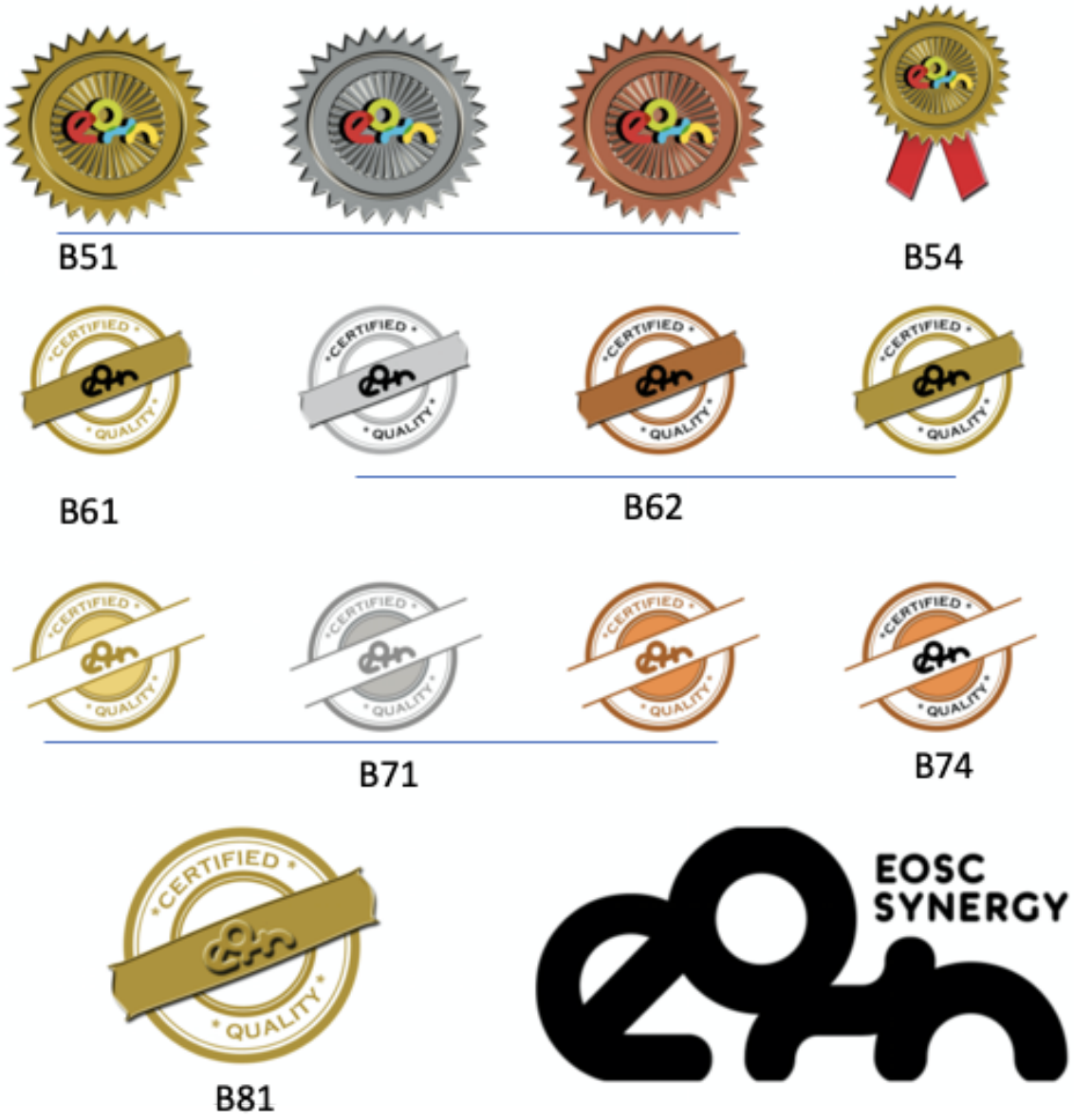

Figure AIII-2: Complete set of the designed digital badges considered by the project.